MDA-Based Design Pattern Components

Liliana Martinez, INTIA - Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil - Argentina, lmartine@exa.unicen.edu.ar

Liliana Favre, INTIA - Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil - Argentina, & CIC, Buenos Aires, lfavre@exa.unicen.edu.ar

ABSTRACT

The Model Driven Architecture (MDA) is an initiative of the Object Management Group (OMG) that promotes the use of models for developing software systems. It distinguishes at least three kinds of models: Platform Independent Model (PIM), Platform Specific Model (PSM) and Implementation Specific Model (ISM). The concepts of models, metamodels and model transformations are at the core of MDA. In this paper we describe a meta-modeling technique to define design pattern components from a MDA perspective. In this context, we propose a “megamodel” for defining reusable components that integrates different kinds of models with their respective metamodels. We analyze metamodel-based model transformations among levels of PIMs, PSMs and ISMs. We illustrate the approach to define reusable design pattern components using the Observer pattern.

1 INTRODUCTION

The Model Driven Architecture (MDA) is an initiative of the Object Management Group (OMG) that promotes the use of models and model transformations for developing software systems (MDA, 2005). MDA distinguishes at least three kinds of models: Platform Independent Model (PIM), Platform Specific Model (PSM) and Implementation Specific Model (ISM). A PIM is a model that contains no reference to the platforms that are used to realize it. A PSM describes a system in the terms of the final implementation platform e.g., .NET or J2EE. An ISM refers to components and applications.

A model driven development is carried out as a sequence of model transformations that includes at least the following steps: construct a PIM; transform the PIM into one or more PSMs, and construct executable components and applications directly from the PSMs (Kleppe et al., 2003; MDA, 2005).

Metamodeling has become an essential technique to support model transformations. In MDA metamodels are expressed using MOF (Meta Object Facility) that defines a common way for capturing all the standard and interchange constructs. They are expressed as a combination of UML class diagrams and OCL constraints (UML, 2005; OCL, 2005). The 4 main core metamodeling constructs are classes, binary associations, data types and package.

MDA is a young approach and several technical issues are not adequately addressed. The success of MDA depends on the definition of model transformations and component libraries which make a significant impact on MDA-based tools. To date existing CASE tools (CASE TOOLS, 2005) do not provide adequate support to deal with component-based reuse and MDA.

In this paper we analyze a technique to reach a high level of reusability and adaptability of MDA design pattern components. Design pattern describes solutions to recurring design problems. Arnout (2004) analyzes the popular Gamma’s design patterns (Gamma et al., 1995) to identify which ones can become reusable components in an Eiffel library. Their work hypothesis is that “design patterns are good, but the components are better” because they are reusable in terms of code. Our work takes up these ideas and contributes a metamodeling technique to built reusable design pattern components in a MDA perspective.

We propose a “megamodel” to define families of design pattern components in a way that fits MDA. A “megamodel” is a set of elements that represent and/or refer to UML-based models and metamodels at different levels of abstraction (Bezivin et al., 2004). Design Pattern components are described at three different abstraction levels: Platform Independent Component Model (PICM), Platform Specific Component Model (PSCM) and Implementation Component Model (ICM). The subcomponents in the different levels include model and metamodel specifications and their interrelations. Metamodels allow defining as many components as different pattern solutions can appear in a concise way. We analyze metamodel-based model transformations of both PIMs into PSMs, and PSMs into ISMs. We illustrate the approach by using the Observer pattern.

This paper is organized as follows. Section 2 describes a megamodel for defining design pattern components. Section 3 describes how to specify components at the metamodel level. Section 4 shows how to specify model transformations as OCL contracts. Section 5 deals with the related work and compares our approach with other existing ones. Finally, Section 6 considers conclusions and future work.

2 A “MEGAMODEL” FOR DEFINING DESIGN PATTERN COMPONENTS

Most current approaches to reusability in the context of MDA are based on empirical methods focusing on reuse of code models; however the most effective forms of reuse are generally found at more abstract levels of design. Reusability is difficult because it requires taking many different requirements into account, some of them are abstract and conceptual, whereas others such as efficiency are concrete. A good approach for reusability must reconcile them. This work proposes an approach for defining reusable design pattern components that integrate high level specifications, which are independent of any implementation technology, specifications targeted at different platforms and implementations.

To define reusable components we propose a “megamodel” that integrates PIMs, PSMs and code with their respective metamodels. Fig. 1 shows the different correspondences that may be held between several models and metamodels.

We define MDA components at three different levels of abstraction: Platform Independent Component Model (PICM), Platform Specific Component Model (PSCM) and Implementation Component Model (ICM). The PICM includes a UML/OCL metamodel that describes a family of all those PIMs that are instances of the metamodel. A PICM-metamodel is related to more than one PSCM-metamodels. The PSCM includes UML/OCL metamodels that are linked to specific platforms and...
3 SPECIFYING DESIGN PATTERN COMPONENTS

A design pattern describes a family of solutions for recurring design problems. In this section we analyze the Observer pattern (Gamma et al., 1995). We present a description of the Observer Pattern, a PIM-metamodel and a PSM-metamodel based on an Eiffel platform.

3.1 Observer Pattern Description

The design pattern Observer "defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically" (Gamma et al., 1995, p. 293).

- **Subject**: Any number of observer objects may observe a subject. It keeps a collection of observers and provides an interface for adding and removing observer objects (Attach and Detach). Whenever its state changes (typically the values of some of its attributes change), the subject will notify its observers (notify).
- **Observer**: It can observe one or more subjects. It defines an updating interface for objects that should be notified of changes in a subject (update).
- **Concrete Subject** (ClockTimer): It stores state of interest to ConcreteObserver objects and sends a notification to its observers when its state changes.

3.2 PIM-Metamodel of the Observer Pattern

The Observer pattern metamodel at PIM level specifies the structural and interaction view of pattern solutions. PIM-metamodels describe all the concepts of the structural and interaction view of pattern solutions. The PSCM level includes metamodels for particular platforms and, the ICM level includes metamodels for different programming languages.

**Concrete Observer** (AnalogClock and DigitalClock): It maintains a reference to a ConcreteSubject object. It stores state that should stay consistent with the subjects and implements the observer updating interface to keep its state consistent with the subjects (Update).

Fig. 2 shows a UML class diagram and a UML sequence diagram of a typical application using Observer pattern.

Semantics

**AbstractObserver**. This metaclass specifies the characteristics of Observer class inside the Observer pattern. It should have at least one operation with the characteristics of Update. Each instance of this metaclass can be an abstract class or an interface. If the instance is an abstract class, a concrete observer inherits its behavior, therefore there is an inheritance relationship with the concrete observer. If the instance is an interface, there is a realization relationship with the concrete observer.

**ConcreteObserver**. This metaclass specifies the characteristics of a concrete observer. It knows the subject (or the subjects), then it is associated to ConcreteSubject through a unidirectional association navigable away from that end.

**AbstractSubject**. Each instance of this metaclass can be an abstract class or an interface and it has at least three operations specified by Attach, Detach and Notify. If the instance of this metaclass is an abstract...
class, all concrete subjects inherit its behavior, therefore there is an inheritance relationship with the concrete subject.

If the instance is an interface, there is a realization relationship with the concrete subject. If the instance of AbstractSubject is an abstract class, it is associated to an instance of AbstractObserver through a unidirectional association navigable away from that end.

ConcreteSubject. This metaclass specifies the characteristics of a concrete subject. It has at least two operations specified by GetState and SetState and its internal state is specified by the ObserverState metaclass.

3. 3 PSM-Metamodel of the OBSERVER Pattern

For each design pattern at the PIM level there are a number of metamodels corresponding to different platforms. The metamodel of the observer pattern in an EIFFEL platform is described below. Fig. 5 partially shows the UML specialized metamodel of the Observer pattern in the Eiffel platform (structural view) and some OCL rules. The shaded metaclasses correspond to metaclasses of the Eiffel metamodel.

4 SPECIFYING METAMODEL-BASED MODEL TRANSFORMATIONS

A model transformation is a specification of a mechanism to convert the elements of a model, that are instances of a particular metamodel, into elements of another model which can be instances of the same or different metamodels. Metamodel transformations are a specific type of model transformations that impose relations between pairs of metamodels. They can be used in the specification stages of the MDA-based developments to check the validity of a transformation.

We specify metamodel-based model transformations as OCL contracts that are described by means of a transformation name, parameters, local operations, preconditions and postconditions. The precondition states the conditions that must hold whenever the transformation is applied. The postcondition states the properties that the transformation guarantees when it was applied.

Next, we partially exemplify a transformation from a PIM to an Eiffel-based PSM that is included in an Observer Pattern component. The definition of the transformation from PIM to PSM uses both the specialized UML metamodel of the Observer pattern and the UML metamodel of an Eiffel platform as source and target parameters respectively. The source metamodel describes a family of packages whose elements are only classes and associations. The postconditions establish correspondences among classes, their superclasses, parameters, operations, and associations. The transformation specification guarantees, for instance, that for class sourceClass in the source model exists a class targetClass in the target model, both of them with the same name, the same parent classes, the same child classes and so on. (See Box A.)

5 RELATED WORK AND DISCUSSION

In Budinsky et al. (1996) a tool to automatically generate code of design patterns from a small amount of information given by the user is described. This approach has two widespread problems. The user should understand “what to cut” and “where to paste” and both cannot be obvious. Once the user has incorporated pattern code in his application, any change that implies to generate the code again will force it to reinstate the pattern code in the application. The user cannot see changes in the generated code through the tool.

Florijn et al. (1997) describes a tool prototype that supports design pattern during the development or maintenance of object-oriented programs.

Albin-Amiot & Guéhéneuc (2001a) describes how a metamodel can be used to obtain a representation of design patterns and how this representation allows both automatic generation and detection of design patterns. The contribution of this proposal is the definition of design patterns as entities of modeling of first class. The main limitation of this
Figure 5. Observer pattern metamodel - EIFFEL platform

Box A.

Transformation PIM-UML TO PSM-EIFFEL

parameters
sourceModel: Design Pattern Metamodel::Package

targetModel: Design Pattern Metamodel-EIFFEL::Project

local operations
equivalentType (a_type: Design Pattern Metamodel::Classifier,
    another_type: Design Pattern Metamodel-EIFFEL::Classifier): Boolean
... 

pre:
sourceModel.importedElement { o is Empty }

post:
sourcetypeModel.ownedElement { o select (oclIsTypeOf(Class)) o size() } = 
sourceModel.importedElement { o select (oclIsTypeOf(Interface)) o size() } = 
targetModel.ownedElement { o select (oclIsTypeOf(EiffelClass)) o size() } = 

post:
sourcetypeModel.ownedElement { o select (oclIsTypeOf(Class)) o size() } forAll (sourceClass / 
    targetClass) exists (sourceClass.name = targetClass.name and 
    sourceClass.generalization.paren = targetClass.generalization.paren and 
    sourceClass.specialization.child = targetClass.specialization.child and 
    sourceClass.templateParameter = targetClass.templateParameter and 
    sourceClass.feature { o select (oclIsTypeOf(Attribute)) o size() } forAll (sourceAtt / 
    targetAtt) exists (sourceAtt.name = targetAtt.name and 
    sourceAtt.visibility = targetAtt.visibility and 
    equivalentType (sourceAtt.type, targetAtt.type) ) ) 

and
sourceClass.feature { o select (oclIsTypeOf(Operation)) o size() } forAll (sourceOp / 
    targetOp) exists (targetOp.name = sourceOp.name and 
    equivalentType (sourceOp.type, targetOp.type) and 
    sourceOp.parameters { o select (oclIsTypeOf(Parameter)) o size() } = 
    targetOp.parameter { o select (oclIsTypeOf(Parameter)) o size() } = 
    Sequence { i | sourceOp.parameter[i].name.size() = 
    targetOp.parameters[i].name.size() = 
    equivalentType (sourceOp.parameter[i].type, targetOp.parameters[i].type) } ) |
... 

6 CONCLUSIONS AND FUTURE WORK

In this paper we analyze a metamodeling technique to reach a high level of reusability and adaptability of MDA-based design pattern components. We propose a megamodel for defining components that integrates PIMs, PSMs, and code models with their respective metamodels. We use our approach to specify standard design patterns. We are validating the technique through rigorous forward engineering processes based on design patterns that integrate formal specifications and UML/OCL (Favre, 2005a).
A crucial problem is how to detect sub-diagrams which can be matched with a pattern. Metamodeling helps in the identification of design patterns by signature matching and semantic matching. The metamodel establishes what elements should be present in the pattern model and their restrictions. Metamodeling allows us to check models against a set of rules to ensure precise and consistent transformations.

REFERENCES
Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
Related Content

Design of a Migrating Crawler Based on a Novel URL Scheduling Mechanism using AHP
[www.irma-international.org/article/design-of-a-migrating-crawler-based-on-a-novel-url-scheduling-mechanism-using-ahp/169176](www.irma-international.org/article/design-of-a-migrating-crawler-based-on-a-novel-url-scheduling-mechanism-using-ahp/169176)

GPU Based Modified HYPR Technique: A Promising Method for Low Dose Imaging
[www.irma-international.org/article/gpu-based-modified-hypr-technique/133532](www.irma-international.org/article/gpu-based-modified-hypr-technique/133532)

A Comparison of Data Exchange Mechanisms for Real-Time Communication
[www.irma-international.org/article/a-comparison-of-data-exchange-mechanisms-for-real-time-communication/186859](www.irma-international.org/article/a-comparison-of-data-exchange-mechanisms-for-real-time-communication/186859)

Challenges for Big Data Security and Privacy
[www.irma-international.org/chapter/challenges-for-big-data-security-and-privacy/183751](www.irma-international.org/chapter/challenges-for-big-data-security-and-privacy/183751)

Schema Evolution
[www.irma-international.org/chapter/schema-evolution/112467](www.irma-international.org/chapter/schema-evolution/112467)